FPGA Microprocessor

Part 2

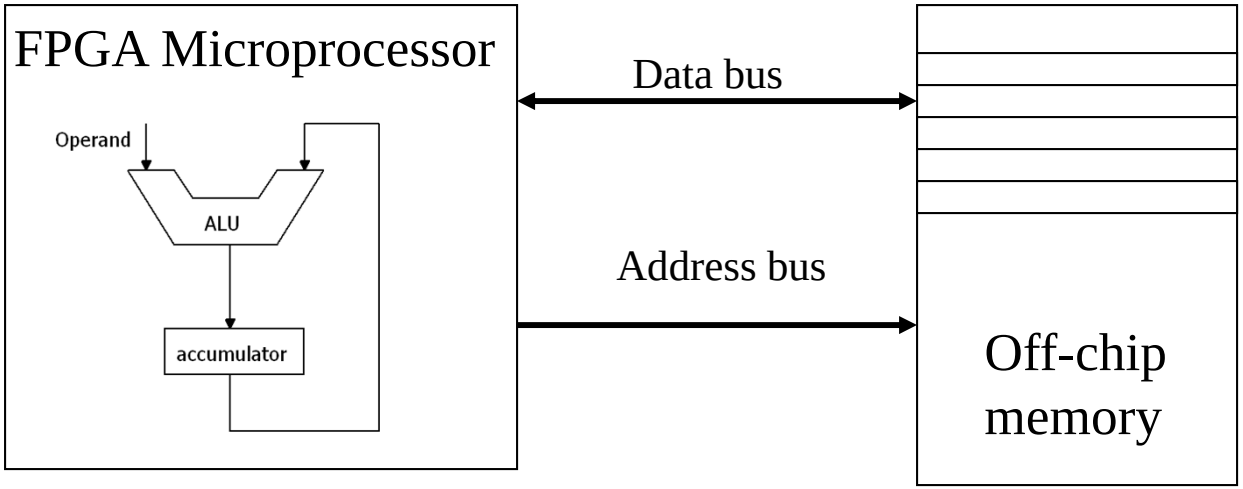
Haibo Wang
ECE Department
Southern Illinois University
Carbondale, IL 62901

How to make FPGA Microprocessor Faster?

- Register File
- Pipeline
  - Example: xr16 FPGA RISC microprocessor
- Cache Memory
- Custom Instructions
  - Custom instruction for computing $A*X^2 + B*X + C$

Register File

- Microprocessor with single register (accumulator)

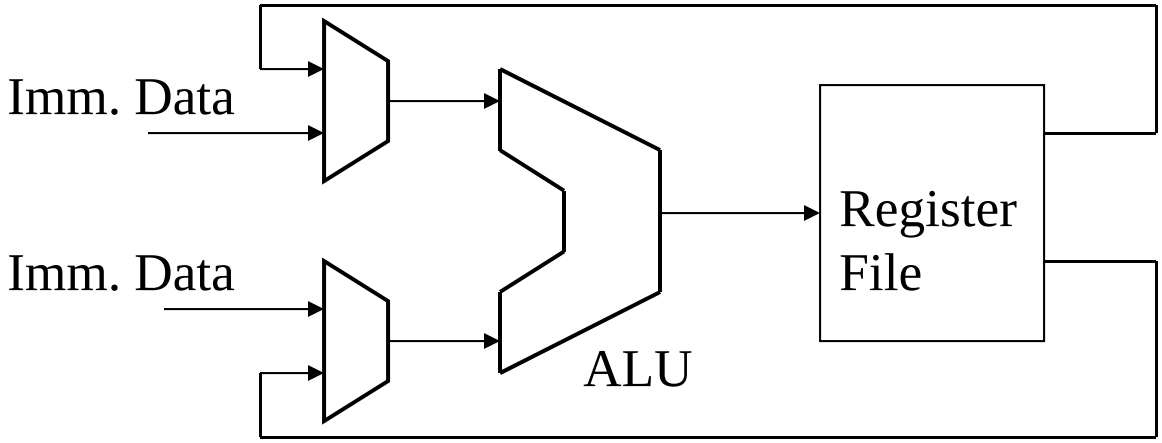


— Disadvantage: Microprocessor has to frequently access off-chip memory

1) Slow
2) Large power consumption
3) Increased memory traffic

Register File

- Microprocessor with multiple registers (register file)



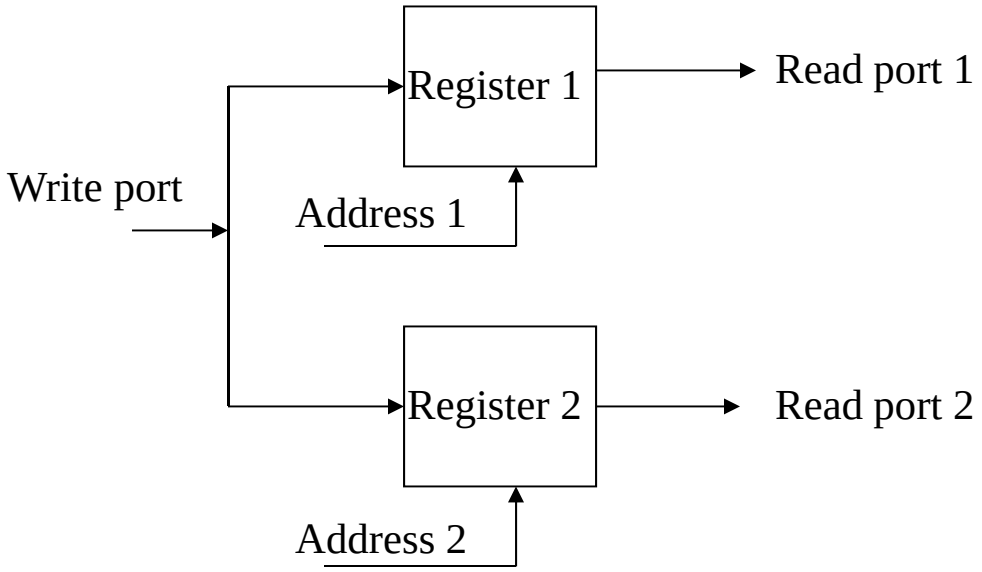
— Advantage: It reduces the frequency that the Microprocessor accesses off-chip memory

1) Increase operation speed
2) Reduce power consumption
3) Reduce memory traffic

— For the above structure, the register file is preferred to have one write port and two read ports.

Register File

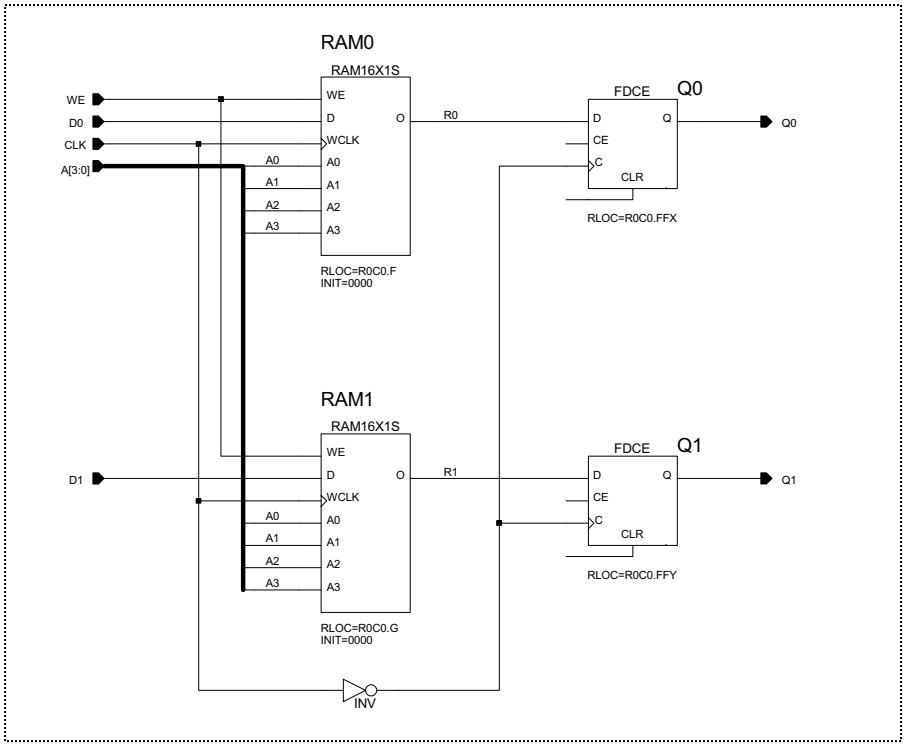
- FPGA Implementation of Register File



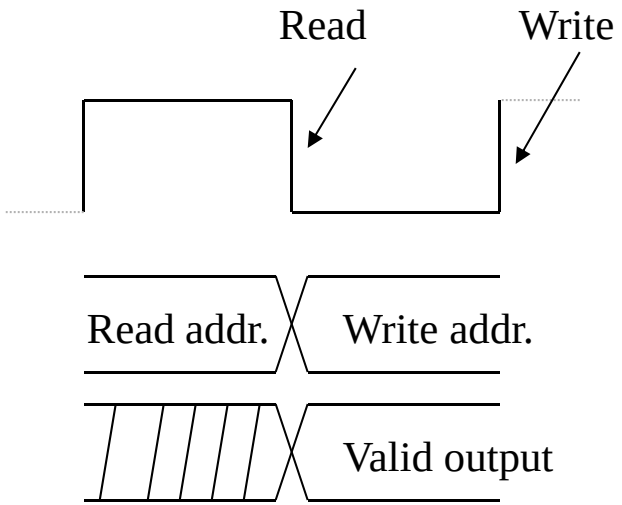
1. During write operation, address 1 and address 2 have the same address and the same data is written into Register 1 and Register 2.
2. Two different memory locations can be read simultaneously by applying different addresses to Register 1 and Register 2

Register File

- Register Implementation on Xilinx FPGAs

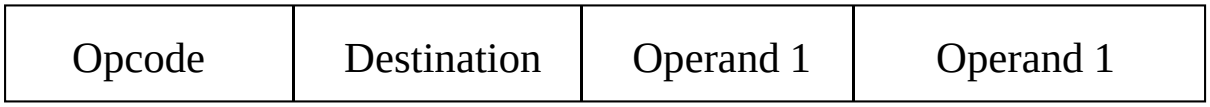


Xilinx XC4000 CLB



Register File

- Instruction format for microprocessors with multiple registers
  - ***Possible Format 1***



Add R1, R2, R3 ⇨ R1 = R2 + R3

  - ***Possible Format 2***

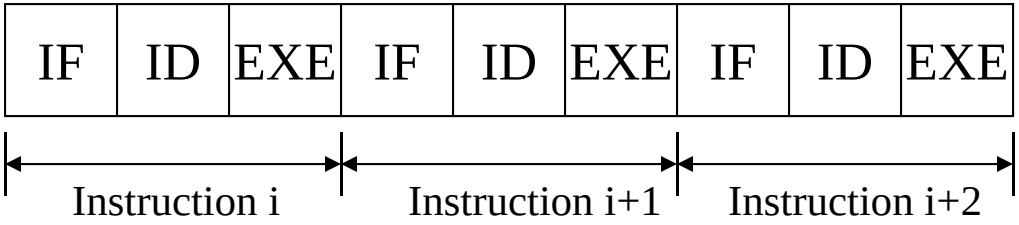
Add R1, R2 ⇨ R1 = R1 + R2

Load R1, [120] ⇨ R1 = Memory (120)

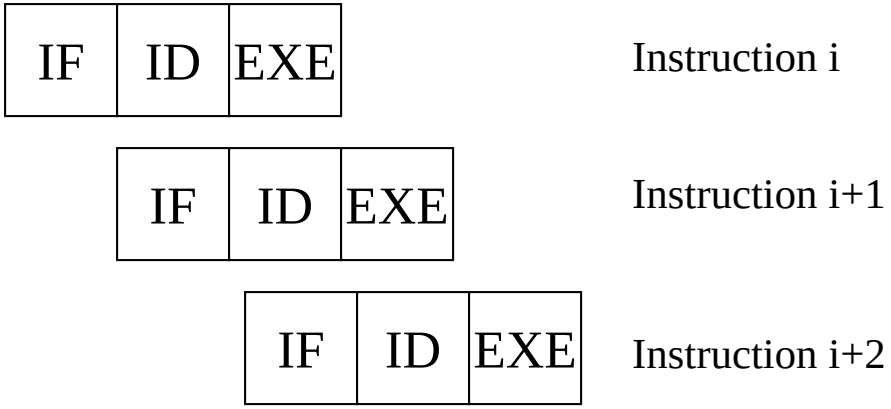
Store [120], R1 ⇨ Memory (120) = R1

Introduction to Pipeline

- Instruction execution without pipeline



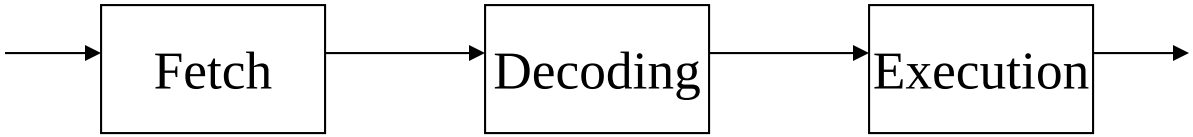
- Instruction execution with pipeline



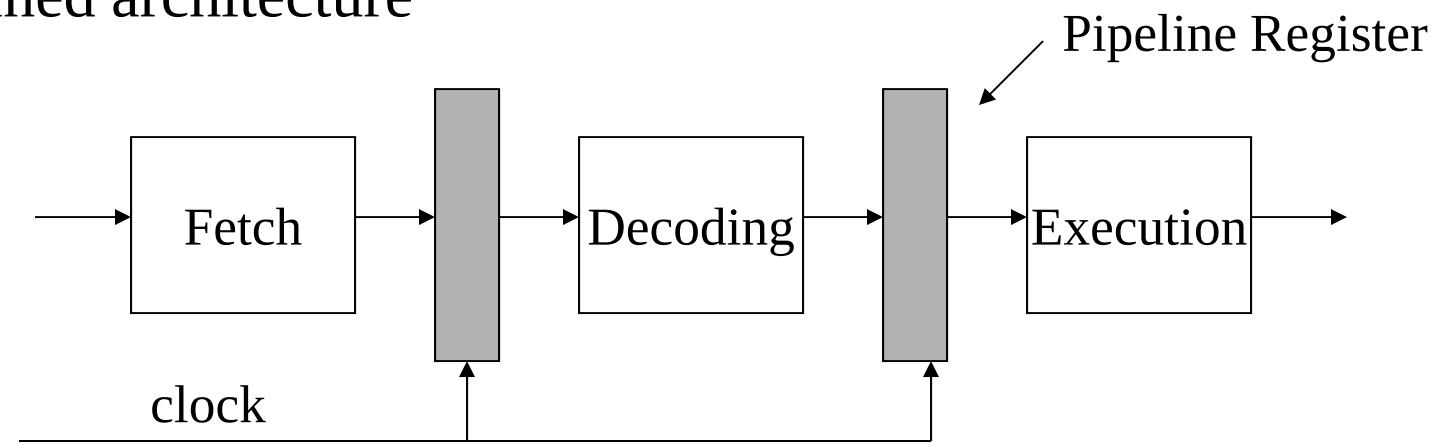
IF: Instruction fetch
ID: Instruction decoding
EXE : Instruction execution

Hardware Implementation

- Non-pipelined architecture



- Pipelined architecture



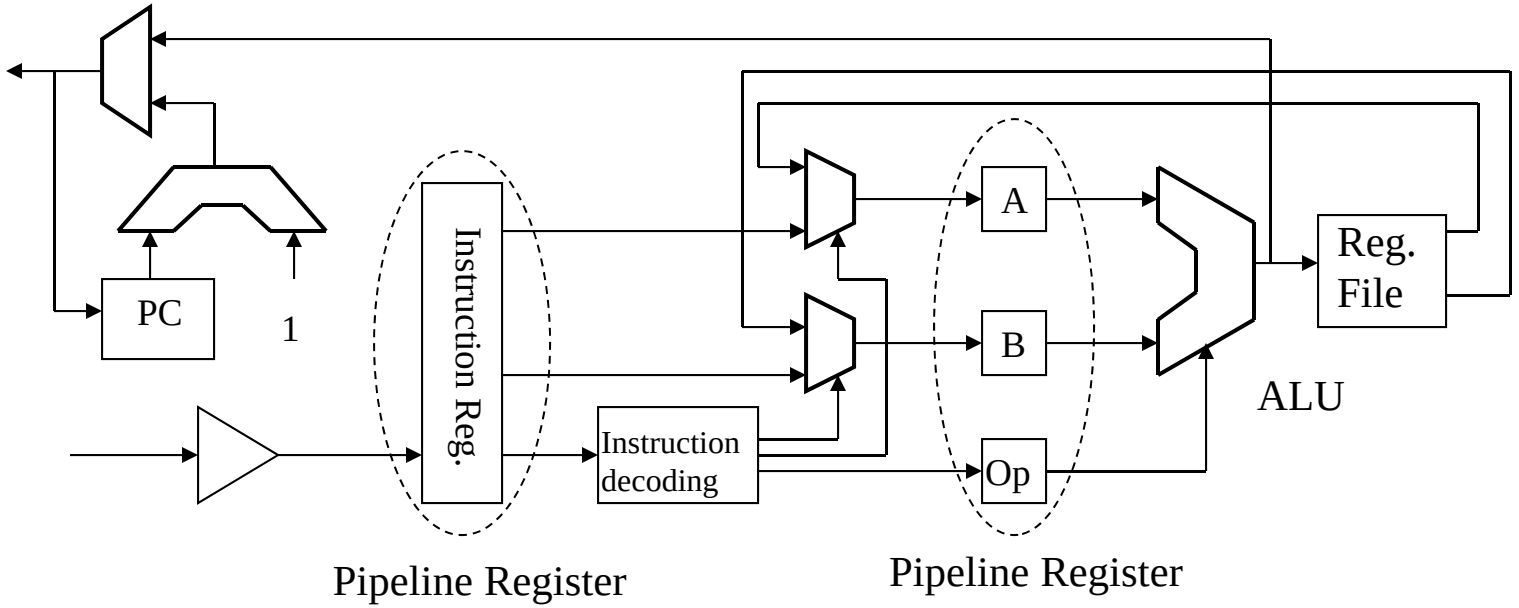
- The register store instructions, operands, and control signals
- The clock frequency is determined by the slowest unit in the above circuit

Hardware Implementation

- Pipeline Stages

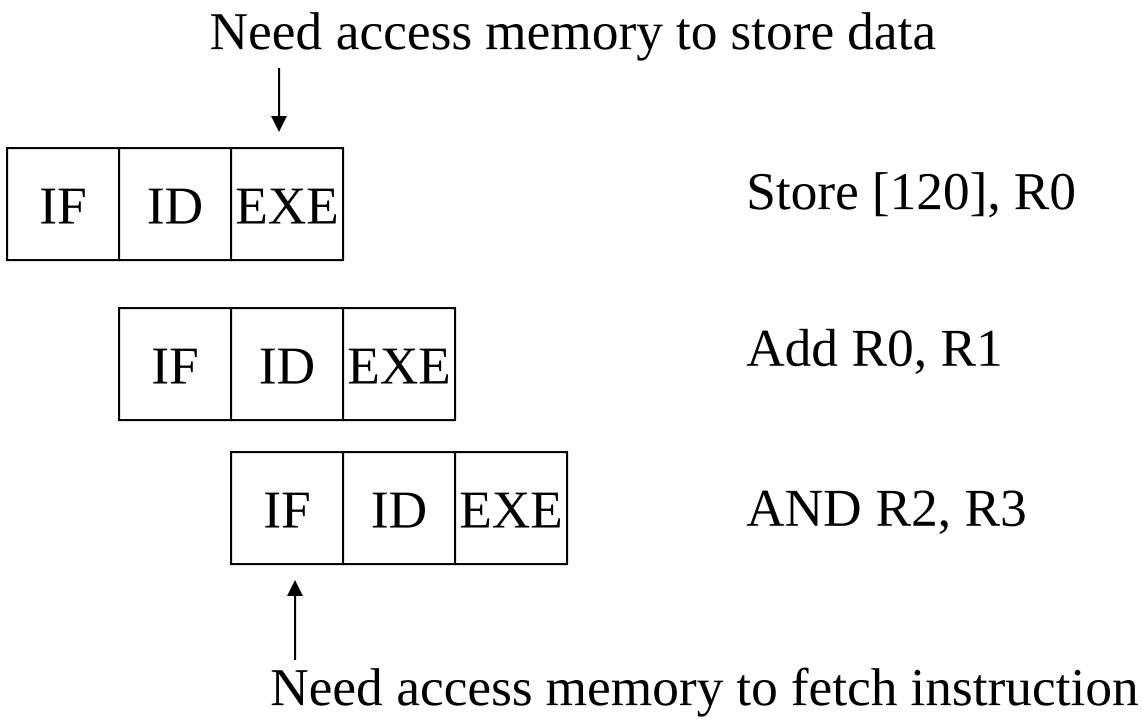
IF	ID	EXE

- Simple hardware Implementation



Structure Hazard

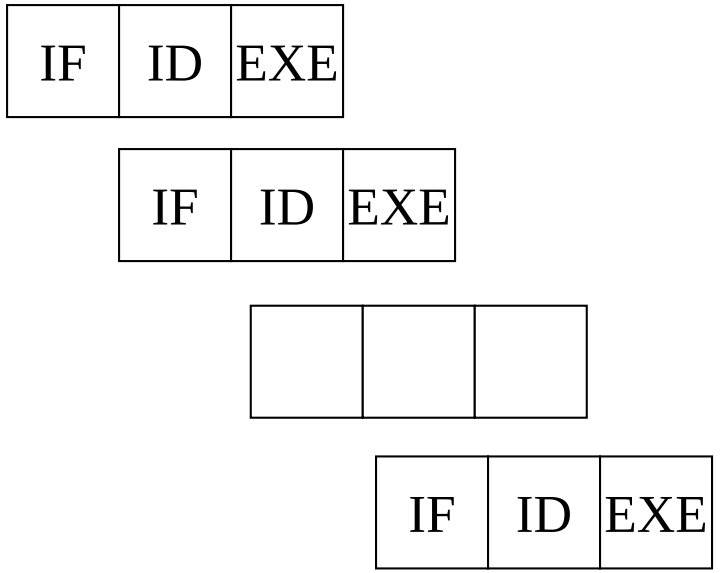
- Structure hazards arise from resource conflicts when hardware cannot support all possible combinations of instructions in simultaneous overlapped execution



Structure Hazard

- **Solution 1:**

  Delay fetching instruction Store R1 by one clock cycle.

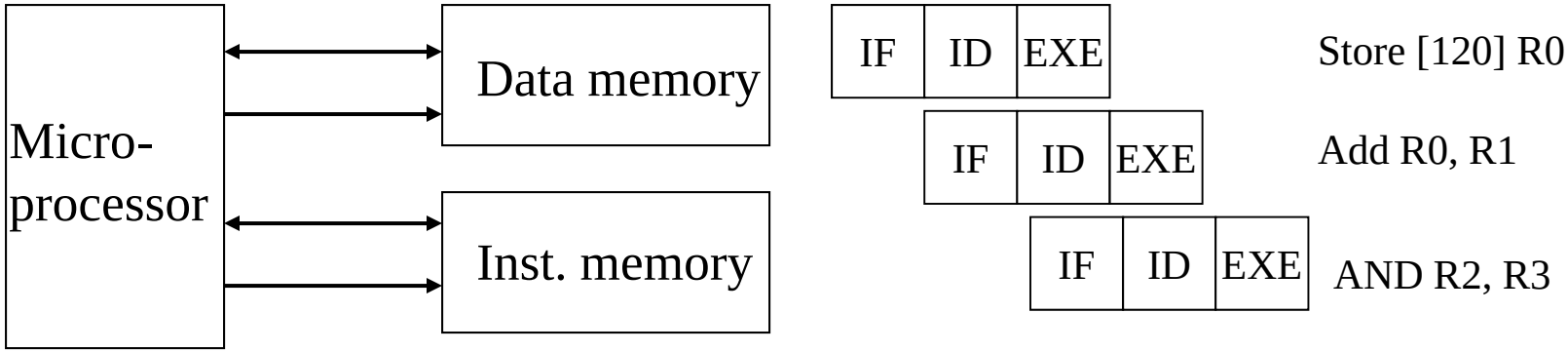


— Advantage: Less expensive to implement.

— Disadvantage: Degrade Performance; need design control circuit to detect resource hazard

Structure Hazard

- **Solution 2:** Use separate memories for data and instructions



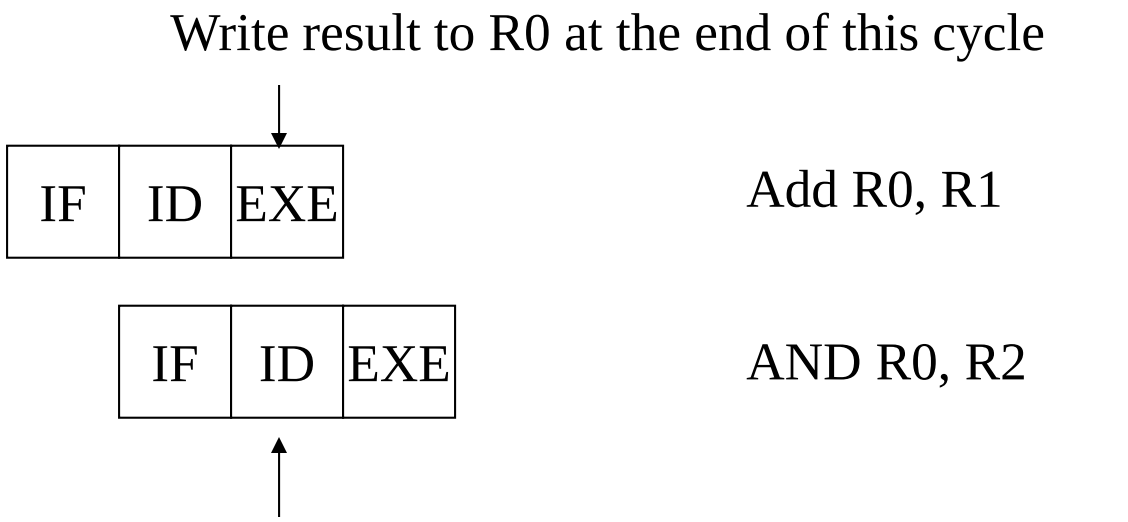
— Disadvantage: Expensive to implement.
— Advantage: Fast performance, less complicated control logic

Note:

1. This solution only alleviates the problem. There still exists resource hazards, *e.g. to execute instructions in the order of Store R1, Add R0.*
2. There are other structure hazards caused by other hardware conflicts.

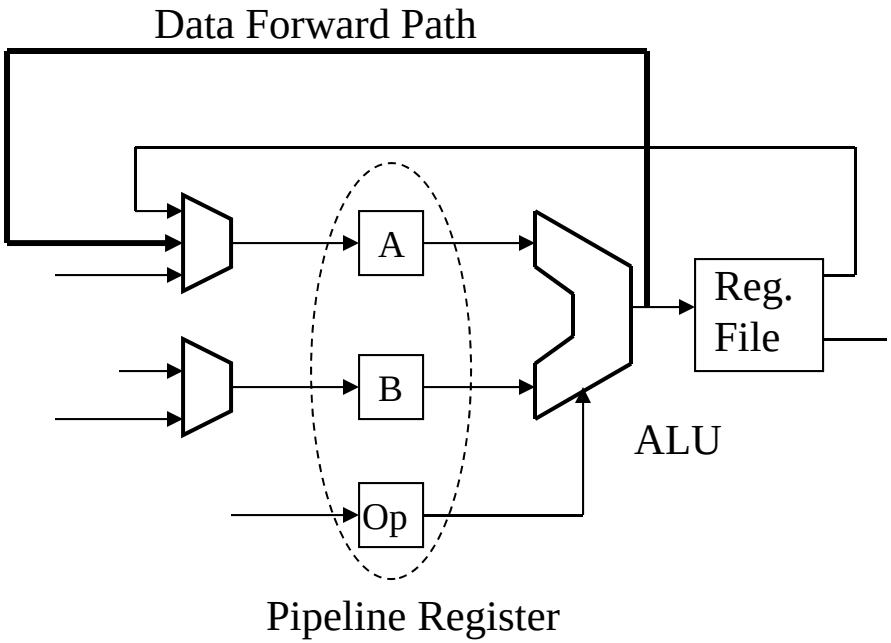
Data Hazard

- Data hazards arise when an instruction depends on the results of a previous instruction. Such hazards are generated if the previous instruction does not generate the results at the time the current instruction needs them.

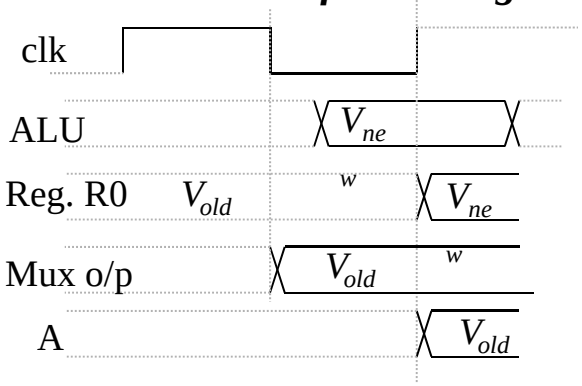


Data Hazard

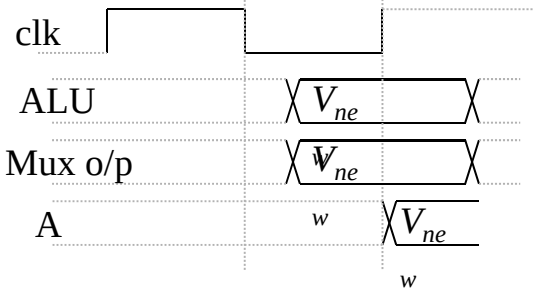
❑ **Solution 1:** Data Forwarding



➢ ***Without data forwarding***



➢ ***With data forwarding***



Add R0, R1

AND R0, R2

Timing diagram for this cycle

Data Hazard

- **Solution 2:** Instruction re-ordering

  - Original Instruction order

    Add R0, R1
    AND R0, R2 ← Data hazard
    Add R5, R6

  - Re-ordered Instructions

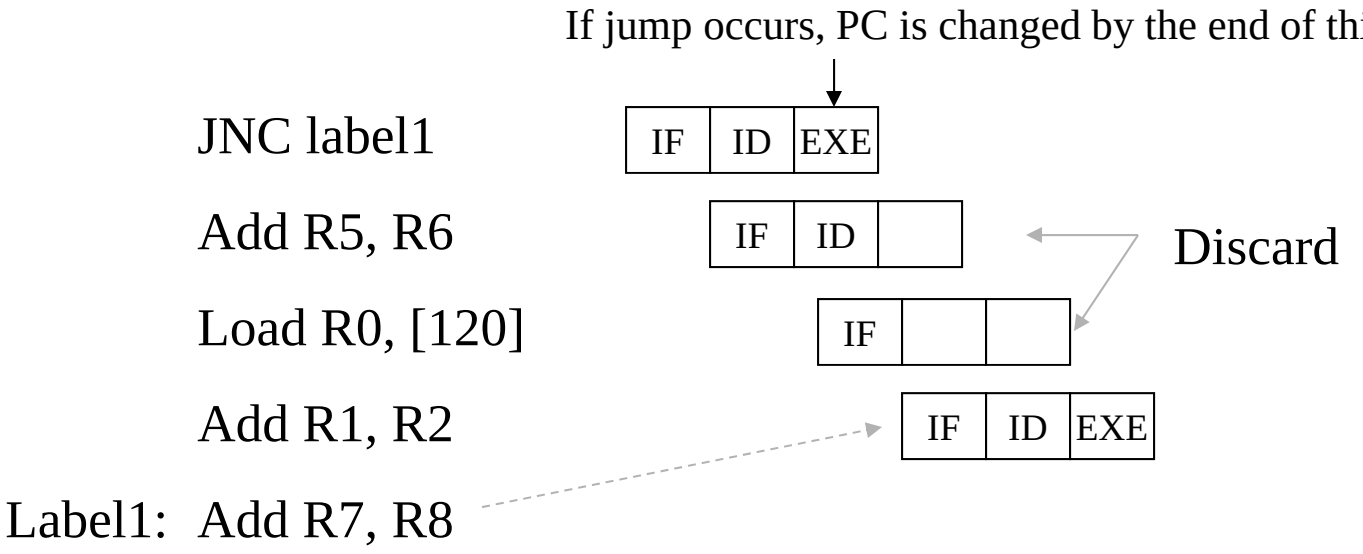
    Add R0, R1
    Add R5, R6
    AND R0, R2 ← No Data hazard

Note:

1. Data forwarding is a hardware-based approach and instruction re-ordering is software-based approach.
2. Even both approaches are used, data hazards can not completely avoided.

Control Hazard

- Control hazards are caused jump and other instructions that change PC value.
  - For the microprocessor shown in slide 11-6, we assume that a jump instruction changes PC register value at its execution cycle.



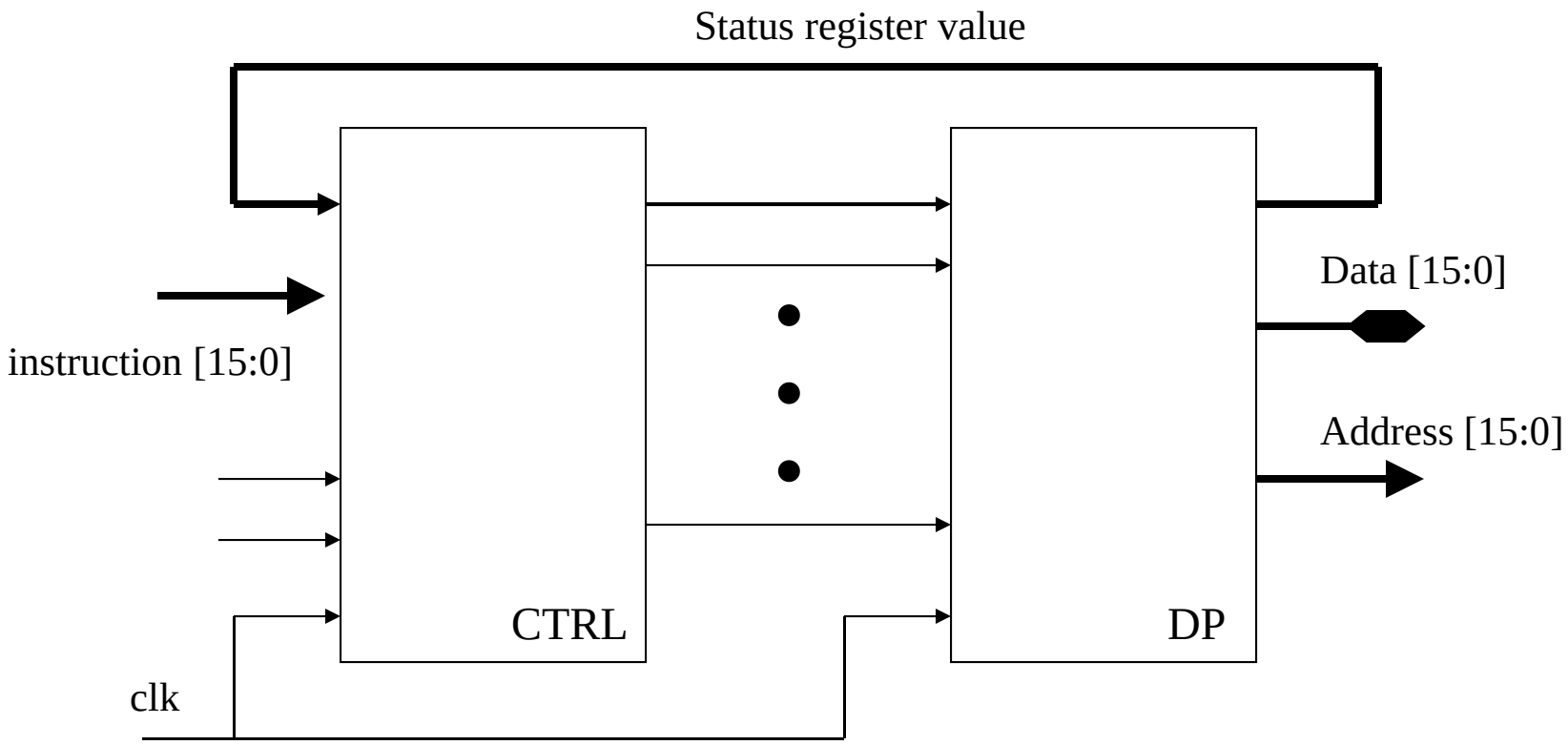
Design Example: xr16 FPGA Microprocessor

- Developed by Jan Gary, Gary Research LLC (*www.fpgacpu.org)*
- RISC Architecture
- 16-bit instructions
- Register file contains 16 16-bit registers
- Load/Store architecture
- Three stage pipeline (IF, ID, EXE)
- Memory is byte addressable

Instructions of xr16 Microprocessor

Hex	Fmt	Assembler	Semantics	N
0*dab*	rrr	`add rd,ra,rb`	rd = ra + rb;	1
1*dab*	rrr	`sub rd,ra,rb`	rd = ra – rb;	1
2*dai*	rri	`addi rd,ra,imm`	rd = ra + imm;	1
3*d***b*	rr	`{and or xor andn adc sbc} rd,rb`	rd = rd *op* rb;	1
4*d***i*	ri	`{andi ori xori andni adci sbci slli slxi srai srli srxi} rd,imm`	rd = rd *op* imm;	1
5*dai*	rri	`lw rd,imm(ra)`	rd = *(int*)(ra+imm);	2
6*dai*	rri	`lb rd,imm(ra)`	rd = *(byte*)(ra+imm);	2
8*dai*	rri	`sw rd,imm(ra)`	*(int*)(ra+imm) = rd;	2
9*dai*	rri	`sb rd,imm(ra)`	*(byte*)(ra+imm) = rd;	2
A*dai*	rri	`jal rd,imm(ra)`	rd = pc, pc = ra + imm;	3
B*dd*	br	`{br brn beq bne bc bnc bv bnv blt bge ble bgt bltu bgeu bleu bgtu} label`	if (*cond*) pc += 2*disp8;	*
C*iii*	i12	`call func`	r15 = pc, pc = imm12<<4;	3
D*iii*	i12	`imm imm12`	imm'next$_{15:4}$ = imm12;	1

xr16 Design Hierarchy



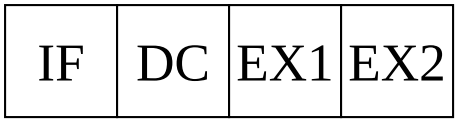
xr16 Pipeline Stages

- IF: Instruction Fetch
  - Fetch instruction
  - Update PC ← PC+2
- DC: Instruction Decoding and Register File Access
  - Decode instructions
  - Read Register operand
- EX: Execute Instruction
  - Perform arithmetic or logic operation
  - Update PC for jump instructions
  - Access memory to perform load or store instructions

Exception for Load/Store Instructions

- A Load or Store instruction need **two execution cycles** to complete

Execution of a load or store



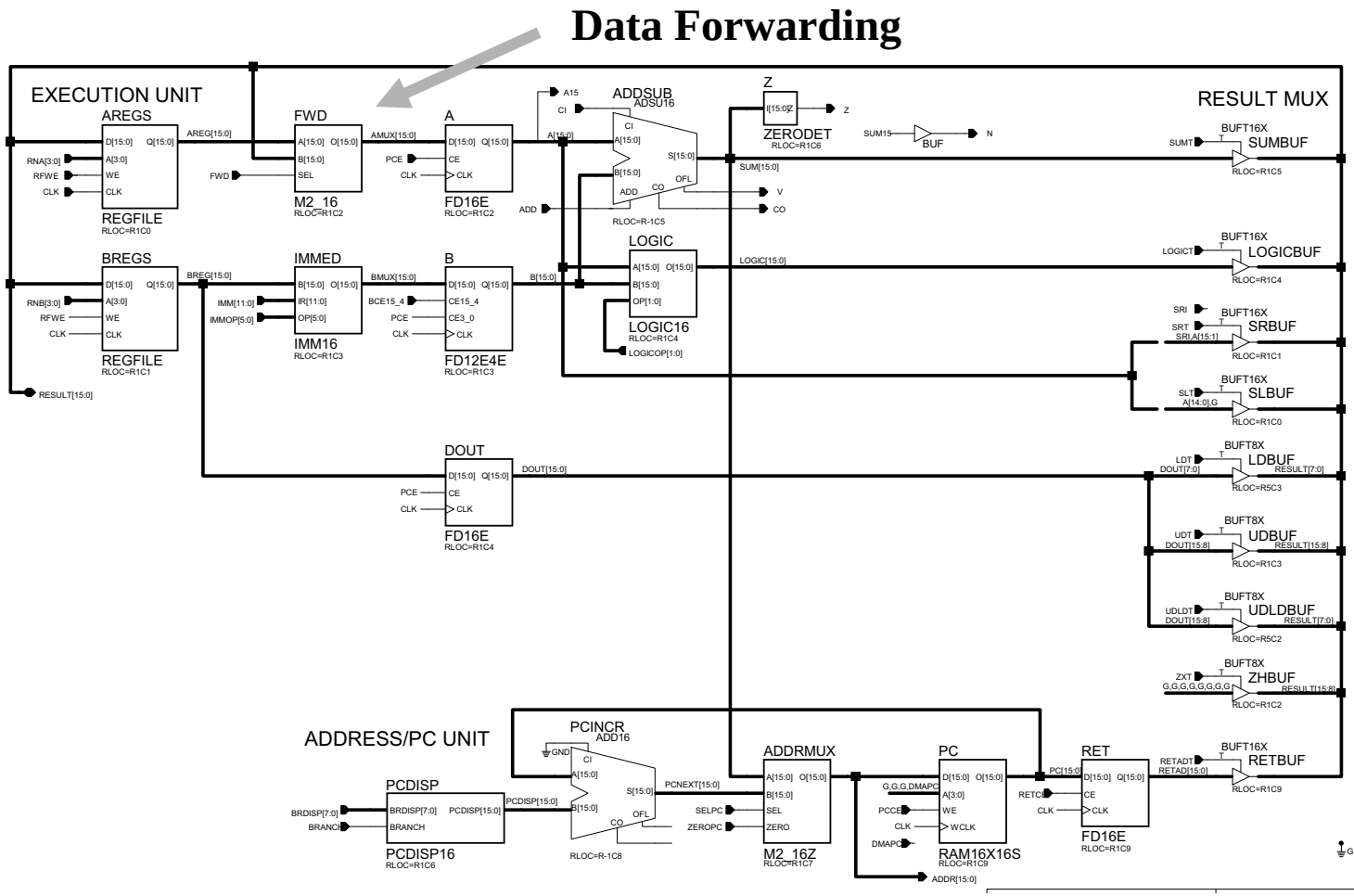
— The execution of Load or Store needs to access memory, which make it longer

— Alternative solution is to slow down clock such that it possible to complete a load or store operation within a clock cycle. However, this solution is not favored because it will significantly slow down the overall performance

Alternative solution

xr16 Pipeline Hazards

- Data Hazards
  - Example: ***ANDi R0, 7, Addi R2, R0, 7***



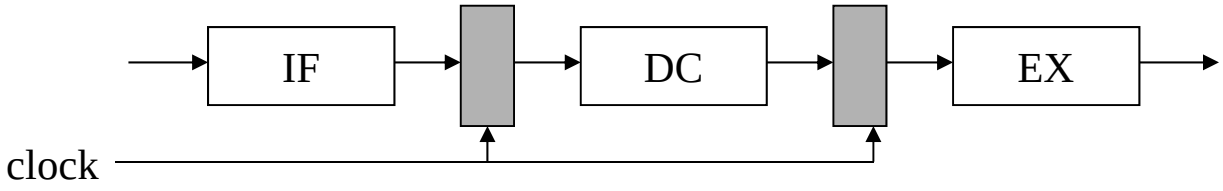
xr16 Pipeline Hazards

- Structure Hazards Caused by Memory Access
  - ✓ Scenario 1: Memory is not ready when fetching the next instruction

t1	t2	t3	t4	t5	t6
IF_1	DC_1	EX_1			
	~~IF_2~~	IF_2	DC_2	EX_2	
			IF_3	DC_3	EX_3

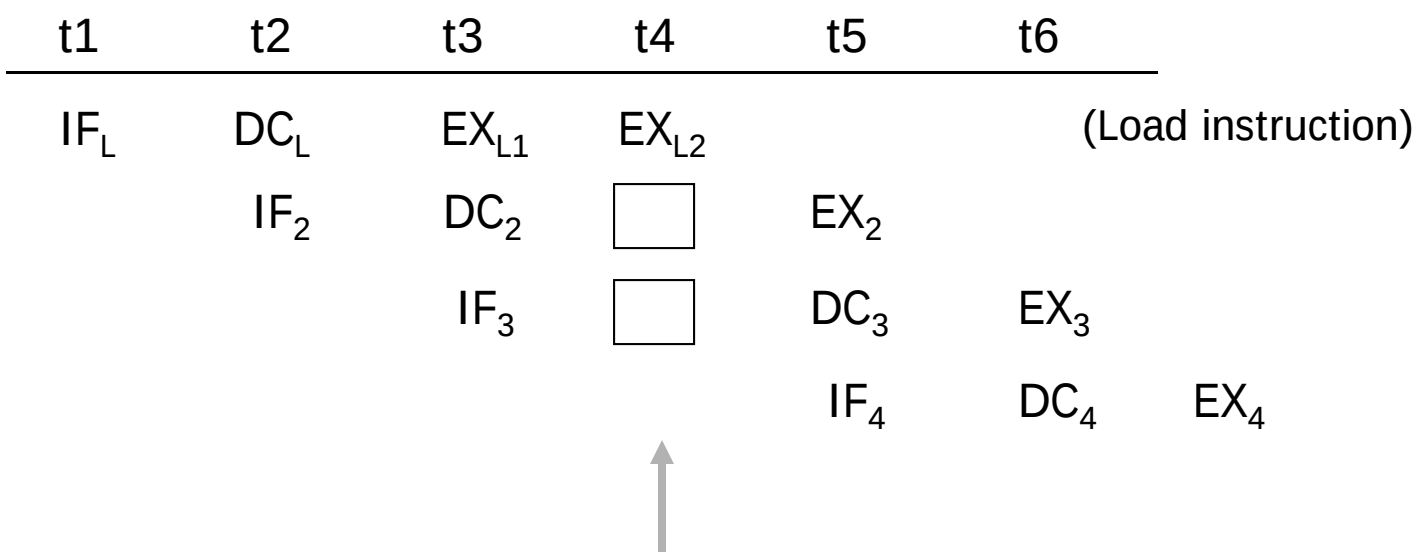
Memory is not ready

Solution: Disable clock that goes to pipeline registers during t3 cycle



xr16 Pipeline Hazards

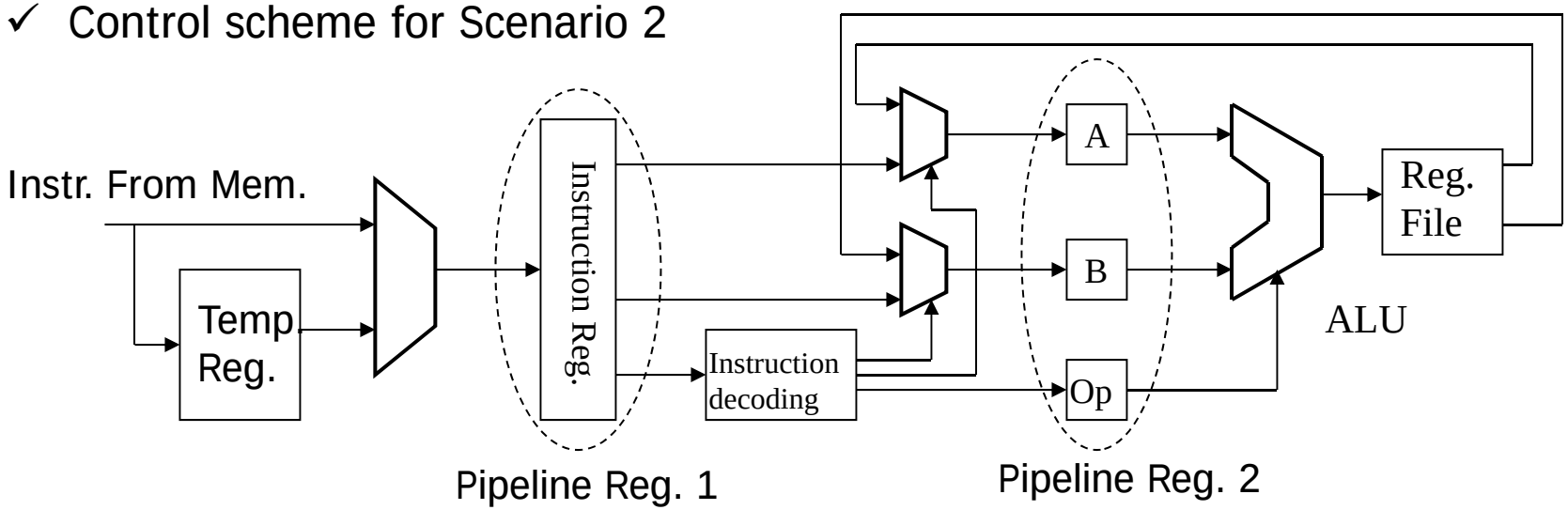
- Structure Hazards Caused by Memory Access
  - ✓ Scenario 2: execution of Load or Store instructions



Load Instruction accesses memory at this clock cycle,
So, new instruction can not be fetched at this clock cycle

xr16 Pipeline Hazards

✓ Control scheme for Scenario 2



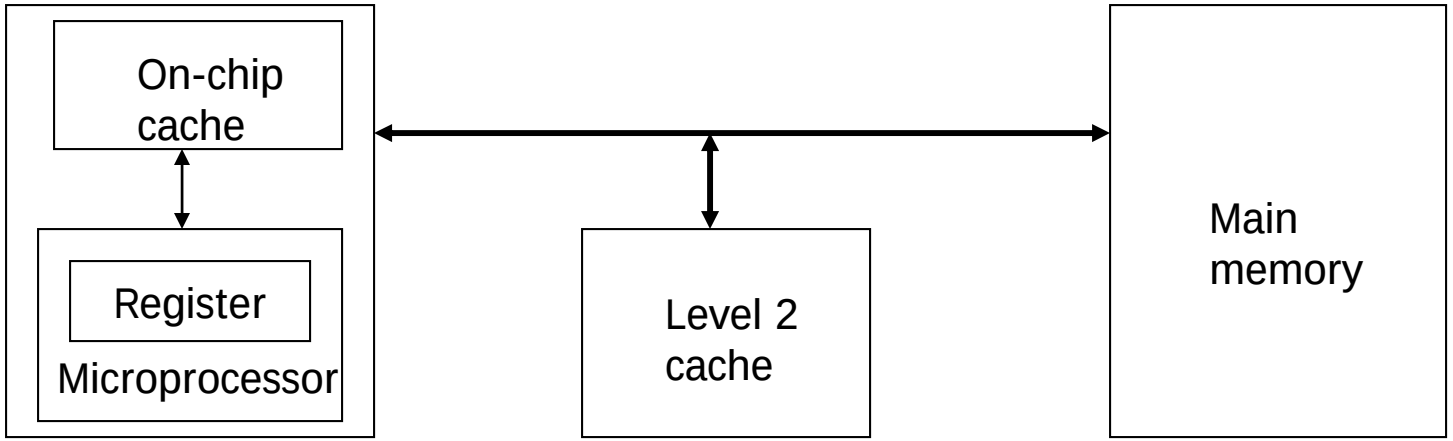
- t3 cycle: Instruction 3 is fetched from memory and stored into Temp. Reg.
- t4 cycle: Pipeline registers remain the same data (by disabling their clock) and complete the Load instruction
- T5 cycle: Fetch instruction 4 from memory (IF_4)
  Load instruction 3 from Temp. Reg. into Pipeline Reg. 1 (DC_3)
  Load operands and ALU op-code into Pipeline Reg. 2 (EX_2)

Introduction to Cache Memory

- Microprocessor speed is normally faster than memory speeds
- Smaller memories are faster than larger memories
- Principle of Locality
  - **Temporal locality:** recently accessed data or instructions are likely to be accessed in the near future
  - **Spatial locality:** items (data or instructions) whose addresses are near close tend to be referenced close together.

Introduction to Cache Memory

- Memory hierarchy

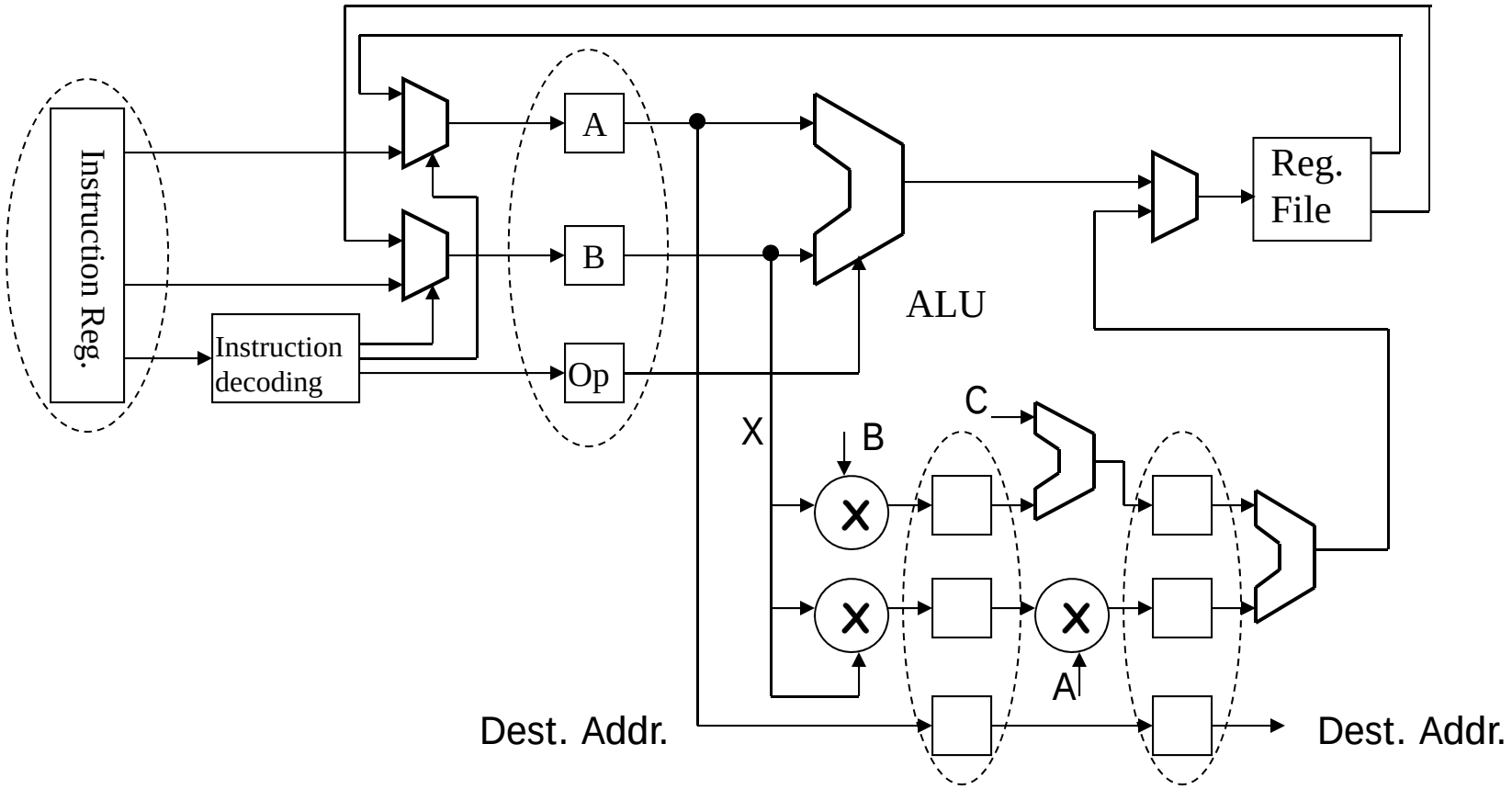


Custom Instructions

- The flexibility of FPGA processors provides another option to improve system performance by implementing custom instructions for critical computations.
  - For example: in an application function $A \bullet X^2 + B \bullet X + C$ is frequently evaluated.
    - If this function is evaluated with a general purpose microprocessor, a small procedure consisting of multiple instructions (such as mul, load, store) need to be executed, which is slow. To improve performance, higher clock frequency is needed
    - If this function is evaluated with an FPGA microprocessor, a custom instruction can be implemented to calculate the function. Only a single instruction is executed to evaluate the function. Even if the FPGA microprocessor has a slower clock frequency, it may still outperform the general purpose microprocessor

Custom Instructions

- Custom Instruction for computing $A \cdot X^2 + B \cdot X + C$



Custom Instructions

- Execution of the Custom Instruction

t1	t2	t3	t4	t5	t6	t7	
IF_1	DC_2	EX_3					(Regular instruction)
	IF_C	DC_C	EX_{C1}	EX_{C2}	EX_{C3}		(Custom instruction)
		IF_3	DC_3	EX_3			
			IF_4	DC_4		EX_4	
				IF_4		DC_4	EX_4

Overview

PicoBlaze™

- 8-bit RISC device
- 256 addressable 8-bit Input ports
- 256 addressable 8-bit Output ports
- 16 GPRs (General Purpose Registers)- No dedicated accumulator register as the ALU (Arithmetic Logic Unit) can be coupled to any GPR.
- 1024 instruction memory (Maximum length)
- 64-byte internal scratch pad RAM- Used to store the contents of any of the 16 GPRs giving more scope for the use of internal variables
- 31 entry stack provides a significant nested 'CALL' depth
- 18-Bit instructions
- Carry and Zero Flags

PicoBlaze Architecture

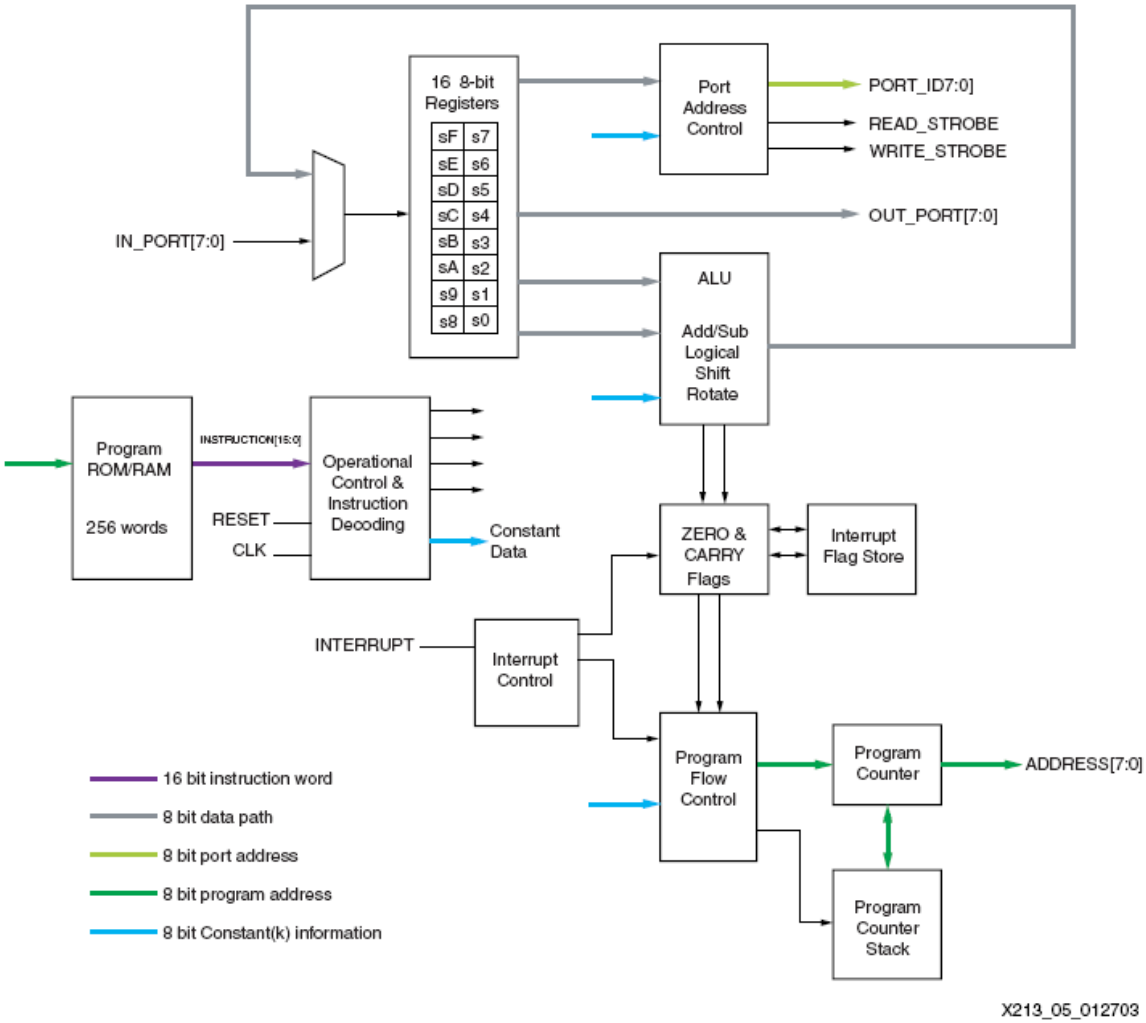


Figure 5: **PicoBlaze Architecture**

PicoBlaze Layout

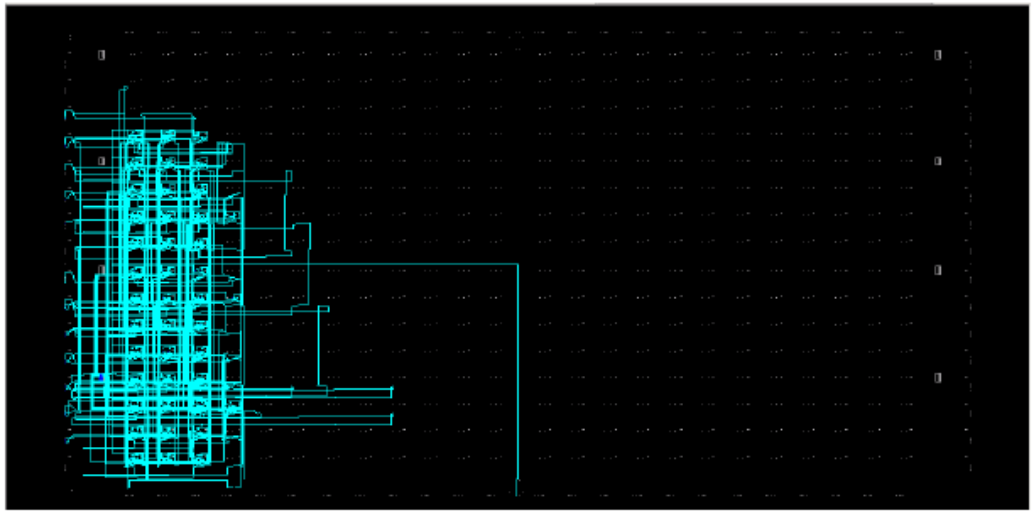


Figure 2: **FPGA Editor View of a PicoBlaze Macro in an XC2S50E Spartan-IIE Device**

XST Report

LUT1	: 1	108 LUTs (54 slices)
LUT2	: 8	
LUT3	: 71	
LUT4	: 28	
muxcy	: 37	Carry and MUX logic (Free with LUTs)
muxf5	: 9	
xorcy	: 28	
FD	: 41	65 Flip_flops (Free with LUTs)
FDE	: 2	
FDR	: 5	
FDRE	: 7	
FDRSE	: 8	
FDS	: 2	
RAM16X1D	: 8	Register bank (8 slices)
RAM16X1S	: 8	Call/Return Stack (4 slices)

Total = 66 Slices

MAP Report

Number of Slices : 76 out of 768 (9%)
Number of Block RAMs : 1 out of 8 (12%)
Total equivalent gate count for design: 19,816

TRACE Report

Device,speed: xc2s50e,-7 (ADVANCED 1.10 2001-12-19)
Minimum period: 12.301 ns
(Maximum frequency: 81.294 MHz)

40.6 MIPS

x213_4_121302

Download PicoBlaze IP

- **http://www.xilinx.com/ipcenter/processor_central/picoblaze/index.htm**

- **Using PicoBlaze in your design**

```
component kcpsm
    Port (      address : out std_logic_vector(7 downto 0);
            instruction : in std_logic_vector(15 downto 0);
                port_id : out std_logic_vector(7 downto 0);
           write_strobe : out std_logic;
               out_port : out std_logic_vector(7 downto 0);
            read_strobe : out std_logic;
                in_port : in std_logic_vector(7 downto 0);
              interrupt : in std_logic;
                  reset : in std_logic;
                    clk : in std_logic);
    end component;
```

Figure 6: **VHDL Component Declaration of KCPSM**

- Instruction sets, see Xilinx Application Notes: XAPP213.pdf

Programming PicoBlaze

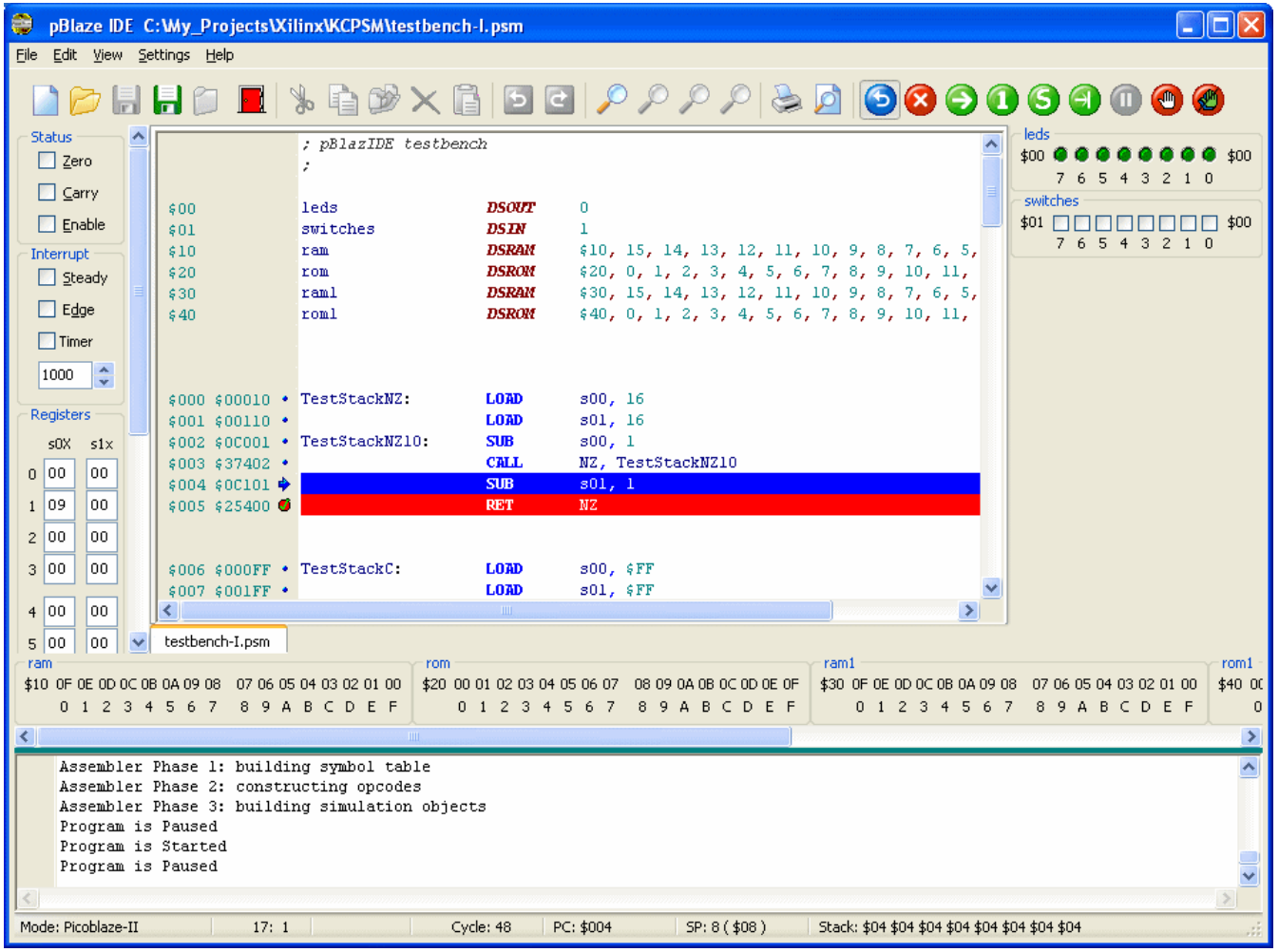
- Programmed using Assembler Language

Class	Command	Flag	Syntax
Program Control	X		JUMP, CALL, RETURN
Arithmetic	X		ADD, ADDCY, SUB, SUBCY, COMPARE
Logical	X		LOAD, AND, OR, XOR, TEST
Shift & Rotate	X		SHIFT, ROTATE
Storage (Scratch Pad)	X		STORE, FETCH
I/O	X		INPUT, OUTPUT
Interrupt	X		RETURNI, ENABLE, DISABLE
Status		X	ZERO, CARRY

PicoBlaze Instruction Set Summary

- Compile the program using KCPSM3
  - The generated HDL ROM code can be synthesized with other design modules.
  - The hex file generated can be loaded via JTAG interface after the circuit is implemented

Simulate your code with pBlazeIDE



http://www.mediatronix.com/pBlazeIDE.htm